Deep Reinforcement Learning Techniques for Solving Hybrid Flow Shop Scheduling Problems: Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C)

Abdulrahman Nahhas
Faculty of computer science
Otto von Guericke University
Magdeburg, Germany
abdulrahman.nahhas@ovgu.de

Andrey Kharitonov
Faculty of computer science
Otto von Guericke University
Magdeburg, Germany
andrey.kharitonov@ovgu.de

Klaus Turowski
Faculty of computer science
Otto von Guericke University
Magdeburg, Germany
klaus.turowski@ovgu.de

Abstract

Well-studied scheduling practices are fundamental for the successful support of core business processes in any manufacturing environment. Particularly, the Hybrid Flow Shop (HFS) scheduling problems are present in many manufacturing environments. The current advances in the field of Deep Reinforcement Learning (DRL) attracted the attention of both practitioners and academics to investigate their adoption beyond synthetic game-like applications. Therefore, we present an approach that is based on DRL techniques in conjunction with a discrete event simulation model to solve a real-world four-stage HFS scheduling problem. The main narrative behind the presented concepts is to expose a DRL agent to a game-like environment using an indirect encoding. Two types of DRL techniques namely, Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C), are evaluated for solving problems of different complexity. The computational results suggest that the DRL agents successfully learn appropriate policies for solving the investigated problem. In addition, the investigation shows that the agent can adjust their policies when we expose them to a different problem. We further evaluate the approach to solving problem instances published in the literature to establish a comparison.

1. Introduction

The presence of scheduling problems is inevitable in the majority of manufacturing environments and many service sectors [1]. Many assembly-production systems such as automotive, printed circuit board production, metal production can be expressed as Hybrid Flow Shop (HFS) production systems. An HFS contains multiple processing stages. HFS scheduling problems constitute a class of problems that falls under the most complicated scheduling problems [2]. At every processing stage, multiple parallel machines are available for processing jobs. All jobs are processed in the same technological order, which is subject to the order of the processing sages. Jobs are processed at most once on every stage through one of the available machines on the processing stage [2].

In HFS production systems, constructing a production schedule includes solving two main sub-problems, namely allocation, and sequencing. The former problem is solved by identifying an optimal or sub-optimal allocation of jobs to the available machines on every processing stage [1]. Many of such scheduling problems can be reduced to well-known classes of bin packing problems [2]. The latter problem is solved by presenting an effective sequence that defines the order, based on which the allocated jobs on every machine are processed. Addressing those two problems combined to construct an effective production schedule is a particularly challenging task. Usually solving HFS scheduling problems is subject to the minimization of the makespan, total tardiness, or setup times [3].

One of the earliest studies on the complexity of scheduling problems is presented by [4, 5]. The latter study shows that the majority of industrial-like scheduling problems are NP-complete [5]. For instance, a rather small number of machines and only two processing stages complicate the problems enough to the degree of NP-completeness [6, 7]. Accordingly, most of the conducted literature analysis on HFS suggests clear dominance in the adoption of heuristic and metaheuristic solution techniques for solving the HFS problems [3, 8, 9]. The main advantage of heuristic techniques is their light execution time that is required to construct a production schedule as for instance most of Priority Dispatching Rules (PDRs) [10]. However, their use is either subject to solving an easy problem or sacrificing the quality of the obtained solutions [11]. In conclusion, heuristic techniques are either problem-specific (specially designed heuristics) or very generic

URI: https://hdl.handle.net/10125/79538
978-0-9981331-5-7

(PDRs). Problem-specific heuristics are effective for solving a well-studied problem and require major modification with the slightest change in the system. On the other hand, conventional metaheuristic optimization techniques such as Genetic Algorithms [12], Simulated Annealing [13], swarm optimization [14], Tabu search, have been widely employed for solving complex HFS scheduling problems [15, 16]. However, despite their high solution quality for solving complex combinatorial optimization problems, they are computationally expensive [7, 17].

To address these concerns, many research efforts on the design and use of hybrid techniques are presented for instance in [15, 18–20]. However, we found limited research on the use of machine learning techniques for addressing scheduling problems of such complexity as we will discuss in the related works section. Nevertheless, some recent applications of machine learning techniques to address HFS scheduling problems are presented in [21]. The authors adopted NeuroEvolution of augmenting topologies to solve a two-stage HFS scheduling problem. The conceptual design of this technique includes the use of Genetic Algorithms (GA), that propose different arbitrary neural networks with different structures. These networks are then employed to estimate the sequence, based on which the jobs are processed on the machines. The investigated problem is subject to the minimization of the makespan and total tardiness.

With the establishment of different technologies in the context of Industry 4.0, real-time data of production systems can be used to suggest better scheduling policies [22]. Yet, the vision of training a machine learning technique constantly over time using real-time data and a simulation environment is promising to support instant decision-making processes. However, such a setup necessitates extensive research efforts on the use of Deep Reinforcement Learning (DRL) for addressing scheduling problems.

Therefore, in this paper, we present an approach based on Deep Reinforcement Learning (DRL) solution techniques and discrete event simulation for solving complex scheduling problems. The investigated problem is subject to the minimization of multi-objective values. The evaluation of the presented concept is based on a real problem in the field of printed circuit board assembly. The formulation of the problem corresponds to the description of the investigated real production system. In the next section, a brief background combined with related works on the adopted DRL solution techniques is presented. The third section presents the conceptual design of the proposed approach. It comprises the formulation of the considered scheduling problem and the encoding of the DRL problem. In the fourth section, we briefly discuss the implementation of the approach and present a thorough analysis of the computational results of the experiments. The paper is concluded in the last section with a brief overview of the limitations and further research directions.

2. Deep reinforcement solution techniques: background and literature analysis

Reinforcement Learning (RL) is a subset of Machine Learning (ML) approaches, that differ from traditional classification, regression, or clustering. RL is a subset of unsupervised ML algorithms in which an instance of the algorithm, called agent, interacts with a specific environment. The environment represents a system or an abstraction over a system, the state of which can be influenced by the agent's actions. More precisely, RL can be described in terms of the Markov Decision Process (MDP) [23]. MDP is represented by $\langle S, A, T, R \rangle$. Within MDP, S represents the set of possible environment states s, transition between which is controlled by the actions of the agent listed within the agent's action space A. The agent picks an action a from the set A at time step t that triggers a transition of the environment to a new state s_{t+1} from the set S. The state transition probability is denoted as $T(s_{t+1}|s_t, a_t)$.

In RL, an additional component is often added to the MDP tuple, this component is discount factor $\gamma \in [0,1]$. Discount factor is a parameter that regulates the strategic behavior of the agent. Values of γ closer to 1 compel the agent to look for higher future cumulative rewards, while potentially taking suboptimal immediate steps. Values of γ closer to 0, instead compel the agent to look for the higher immediate reward at each step. After state transition, the agent receives as input the new state as well as a value of the reward function $R(s_t, a_t, s_{t+1})$, which measures the merit of picking the action a_t within state resulting s_t in transition to s_{t+1}. The correct formulation of the reward function R is one of the most important steps in defining a problem to be solved by RL. It must fully and precisely reflect the expected behavior of the agent. In RL, the agent has no information on what any of the actions in the set A actually represent.

Essentially, the singular objective of the agent in RL is reward maximization. It means that the agent must find the optimal relation between the environment's states and actions. This relation is referred to as policy π, which maximizes the reward taking into account only the current state s_t or communitive reward of taking multiple consecutive actions. Within the context of this work, we concentrate on the scenarios where the agent attempts to maximize the immediate reward. Additionally, in our work, we are specifically focusing

on the RL algorithms based on the application of Deep Neural Networks (DNN) [24]. The subset of these algorithms is generally referred to as Deep Reinforcement Learning (DRL) [25]. In this work, we rely on two algorithms of varying nature from prominent practitioners in the field.

2.1. Proximal Policy Optimization

Proximal Policy Optimization (PPO) [26] is a DRL algorithm developed by prominent industry practitioners at OpenAI. PPO is based on the RL concept of Policy Gradients (PG) [27]. PG is a so-called on-policy RL method because it is attempting to optimize the agent's policies π directly by applying stochastic gradient ascent to the policy parameters. Within the context of DRL, a DNN is employed to facilitate the mapping of actions $a \in A$ to states $s \in S$. One of the notable advantages of PG-based algorithms, including PPO, is the fact that any training data is used once and then discarded. This results in moderate main memory requirements for training and inference.

However, the original PG suffers from the stability of policy learning when too steep adjustments in the policy parameters are made, a challenge described and tacked earlier in [28] and improved upon in [26]. Too steep of the change can result in a policy update that actually is not beneficial but completely ruins that policy, to the point that the agent might never recover. Efficient update of the agent's policies is especially significant in the systems with a high cost of obtaining training samples by making steps in the environment, such as manufacturing scheduling. PPO was developed to efficiently tackle this challenge [26] of making sure that training is efficient, and the most possible policy improvement is done while avoiding changes that can potentially have a catastrophic effect on the policy.

2.2. Asynchronous Advantage Actor-Critic

Asynchronous Advantage Actor-Critic (A3C) [29] is another algorithm developed by prominent practitioners in the industry, specifically Google's DeepMind. As the name of the algorithm suggests it is based on the actor-critic principle, which means it consists out of two core components: actor and critic. The actor is based on the same principles as PG, it's on-policy RL. As in the case of classic PG, based on the current state of the environment, the actor pics an action to $a \in A$ to execute within this environment, in pursue of a higher reward. However, the actor is not receiving a reward directly, instead, the adjustment of the policy is controlled by the critic.

In contrast to the actor, the critic is based not on PG but instead on a different concept called Q-Learning [30]. Here, instead of directly adjusting the policy function parameters, critic estimates Q values. A Q value is an estimation of a reward for a state, action combination, $Q(s, a)$. While Q-Learning as a concept was not originally published with DNN in mind and had significant scale limitations, earlier work of Google's DeepMind, Deep Q-Network (DQN) [31] in the field demonstrated that the use of a DNN for estimating Q values can be done efficiently by employing a DNN. Essentially within this approach, a current state of environment s_t is mapped via DNN to $\langle a, Q \rangle$ for every $a \in A$. The critic receives the action taken by the agent as well as the state and the reward function. Based on this information, the critic evaluates the action and controls the policy adjustment of the actor. Such architecture was demonstrated by the authors to be able to solve complicated problems. Worth noting, A3C is called asynchronous. It means that multiple actor-critics are running at the same time. As stated by the authors, such a constellation is likely to result in a more diverse and thorough exploration of the environment, hence better policies.

2.3. Related Works

The majority of found applications in the field of DRL are evaluated in game-like environments [32]. To the best of our knowledge and based on scanning and searching using Google Scholar, very few applications of DRL for HFS scheduling problems could be found. Nonetheless, we discuss in this section the extracted papers that are found in this field. In [33], the authors presented a reinforcement learning (RL) approach that is supported by a Boltzmann exploration policy to address an HFS problem. The presented encoding is rather evaluated on a small problem instance that included 12 jobs with the single objective to minimize the makespan.

Similarly, in [34], the authors presented an implementation of RL for estimating the value function of Neural Networks (NN) that are used then to map jobs to machines. However, the concept is presented to deal with Pure flow shop scheduling problems that are much simpler than the HFS scheduling problems. Another adoption of RL for real-time decision support is presented in [35]. The authors addressed a three-stage scheduling problem. On all processing stages, two identical machines are available. The authors generated problem instances and observed the performance of the RL approach against EDD, Shortest Processing Time (SPT) and First-In-First-Out (FIFO) rules with respect to the mean flow time, mean lateness, and percentage of late jobs.

As for the adoption of DRL techniques in scheduling, a DQN [31] implementation for addressing

a job shop scheduling problem is presented in [36]. The authors evaluated the presented approach for solving four types of problem instances. The considered problems contain five to twenty jobs that must be scheduled on mostly 5 machines and for one type eight machines. A more sophisticated implementation of DRL to address HFS scheduling problems is presented in [37]. The authors employed PPO to switch positions of two jobs in the sequence and compared their results to some Priority Dispatching Rules (PDR) and Genetic Algorithms (GA). The evaluation of the presented approach is conducted on 26 jobs in the most complicated problem instance that is generated randomly.

3. Problem formulation and presented concept

The investigated production system consists of four production stages. The first three production stages contain five parallel machines of different speeds that are available to process all types of jobs. In the fourth processing stage, two identical machines are available, as presented in Figure 1. Eventually, the associated scheduling activities in this system can be formally formulated as a hybrid flow shop scheduling problem. In addition, the first and fourth production stages are characterized by family major and minor setup times. The second and third processing stages are family-independent stages. It implies that jobs are always scheduled after a minor setup time to reconfigure the machine.

The efficient utilization of such production systems highly depends on the successful minimization of the total number of major setup times required to process a set of jobs. A major setup time is a preparation time that is necessary to reconfigure a machine. This major reconfiguration process is carried out when the machine is switched to process jobs from a different family than the current family. The concept of major and minor setup times is profoundly discussed in [38]. In the investigated production system, the major setup time on the first stage is 65 minutes and the minor is 20 minutes. In the fourth production stage, the major setup time is 120 minutes and the minor setup time is 25 minutes.

3.1. Problem formulation and objective values

In this section, we will shortly present the preliminaries of the considered problem, which is based on the work presented in [20]. A job $j \in J$ is characterized by the following attributes:

- The processing time on the first stage $p_{j,\ s1}$
- The processing time on the second stage $p_{j,\ s2}$
- The processing time on the third stage $p_{j,\ s3}$
- The processing time on the fourth stage $p_{j,\ s4}$
- The due date of a job d_j
- The completion time C_j
- The tardiness of job j is T_j if the completion C_j is bigger than its due date d_j.

Based on the characteristics of jobs, the objective is to find a production schedule, in which all jobs are processed on the required processing stages. This production plan is subject to the minimization of the makespan C_{max}, the total number of the required major setup times on the first and fourth processing stages $MS_{first,\ fourth}$, the total tardiness T and the total number of unit penalties U. The objective values are presented in formula (1), (2), (3), and (4) respectively. The makespan refers to the maximum completion time overall all jobs that must be scheduled. It is essentially, the point in time, at which the last job in this scheduling period is finished. The major $MS_{first,\ fourth}$ is induced on both stages, whenever the machines are reconfigured to process jobs from different families than the current family on this machine.

$$C_{max},\ max\, C_j : \forall J_j\ (j \in \{1, \ldots, n\} \quad (1)$$

$$MS_{first,\ fourth} \in \{0, \ldots, n-1\} \quad (2)$$

$$T = \sum_{j=1}^{n} T_j\ :\ \forall J_j\ (j \in \{1, \ldots, n\}) \quad (3)$$

$$U = \sum_{j=1}^{n} U_j\ :\ \forall J_j\ (j \in \{1, \ldots, n\}) \quad (4)$$

Addressing multi-objective optimality measures is necessary to account for optimizing the system utilization and customer satisfaction through total tardiness and penalties. We assume that the buffer capacities between processing stages are unlimited. Also, all jobs-related data are known and fixed in advance.

3.2. The proposed conceptual design and the formulation of the DRL problem

The conceptual design of the presented implementation contains two main components. The first component allows employing different DRL agents. In this paper, we investigated the performance of well-established DRL techniques: Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C). We are still in the process of integrating and conducting further analysis on other DRL techniques. The second component is a discrete-

event simulation model. We imitated the considered system in a simulation model, which is subject to the characteristics of jobs, the operational constraints, and production procedures of the investigated production environment. The simulation model is the environment, which is exposed to the DRL agent as a game.

3.3. Action space, observation space and reward function.

We attempt to leverage the capabilities of DRL techniques to address scheduling problems. Our conceptual design of the presented application is based on an indirect encoding of the problem. The goal is eventually to present an encoding similar to a video game, in which an agent has access to either one or two controllers and can interact with the environment such as a game. This encoding is motivated by the fact that most of the applications in the field of deep reinforcement learning are conducted in game-like environments [32].

3.3.1. Action space

As presented in Figure 1 and marked with blue color, we investigated two different encodings. In the first encoding, the agent controls the use of different simple allocation algorithms over time during the simulation for solving the allocation part of the problem in the first processing stage. Based on the selection of the agent, families and their associated jobs are reallocated between available machines in the first processing stage over time. The agent has access to six different allocation algorithms that are based on the work presented in [20]. The sequencing part of the problem is solved using a sequencing algorithm presented in [11]. Briefly, the allocation algorithms are all designed to allocate jobs on a family level between machines based on different sorting.

The second encoding gives the agent another controller to interact with the environment. In addition to a set of allocation algorithms, the agent controls a set of six sequencing algorithms on every machine. These sequencing algorithms are replicated based on the work published in [11]. Briefly, some of the sequencing algorithms favor dispatching jobs more based on the Earliest Due Date (EDD) priority dispatching rule. These rules are more conservative and usually act against system utilization since they tend to deliver results with a high number of major setup times on the first and fourth processing stages while avoiding penalties and total tardiness. Some other sequencing algorithms favor the minimization of the makespan and the total number of major setup times. These simple algorithms tend to dispatch jobs from the same family until a family is fully processed before switching to another family. As a result, penalties on the delivery dates are expected. We relied on multi-discrete action space formulation to generate our actions. This type of action space is used when the agents must produce multiple independent discrete actions to interact with the environment [39]. For further discussion on the multi-discrete action space formulation, one can refer to the works presented in [32, 39]. These actions are then used to select different allocation and sequencing algorithms on the first and fourth processing stages over time for solving the scheduling problem. On the second and third processing stages, we relied on the EDD rule for dispatching jobs since they are family-independent processing stages.

3.3.2. Observation space and reward function

The observation space is divided into three major parts as presented in Figure 1. The first part of the observation space is dedicated to describing the entire production plan that results from the set of actions. It implies that the agent receives information about the starting and finishing time of jobs on every processing stage and machine-related information. The aggregation of this information forms a production schedule that includes a time plan, an allocation of jobs to every stage, and the position in the sequence on every machine. The second part of the observation space feeds the agent with key performance indicators on a job level such as the pure total processing time of a job, the total waiting time of a job, the overall flow time of the job in the system, and the tardiness of a job. The last part of the observation space reports the agent with the current relevant key performance indicators of the entire system. This part of the observation space includes the average flow time, the makespan, the total number of major setup times on the first and fourth processing stages, the total tardiness, and the total number of penalties. The majority of those indicators are well-known objective optimality measures that are used for investigating scheduling problems [3, 9].

Finally, the reward function is formulated based on the makespan (1), the total number of major setup times on the first and fourth stages (2), the total tardiness (3), and the total number of penalties (4). After normalization, the reward function is further formulated in (5). The multiplication with a minus is to reformulate the reward function for a minimization problem. We punish the agent further if there are penalties in the solutions by 10. That is to increase the importance of penalties for agents.

$$R = -(C_{max} + MS_{first,\ fourth} + T + U*10) \quad (5)$$

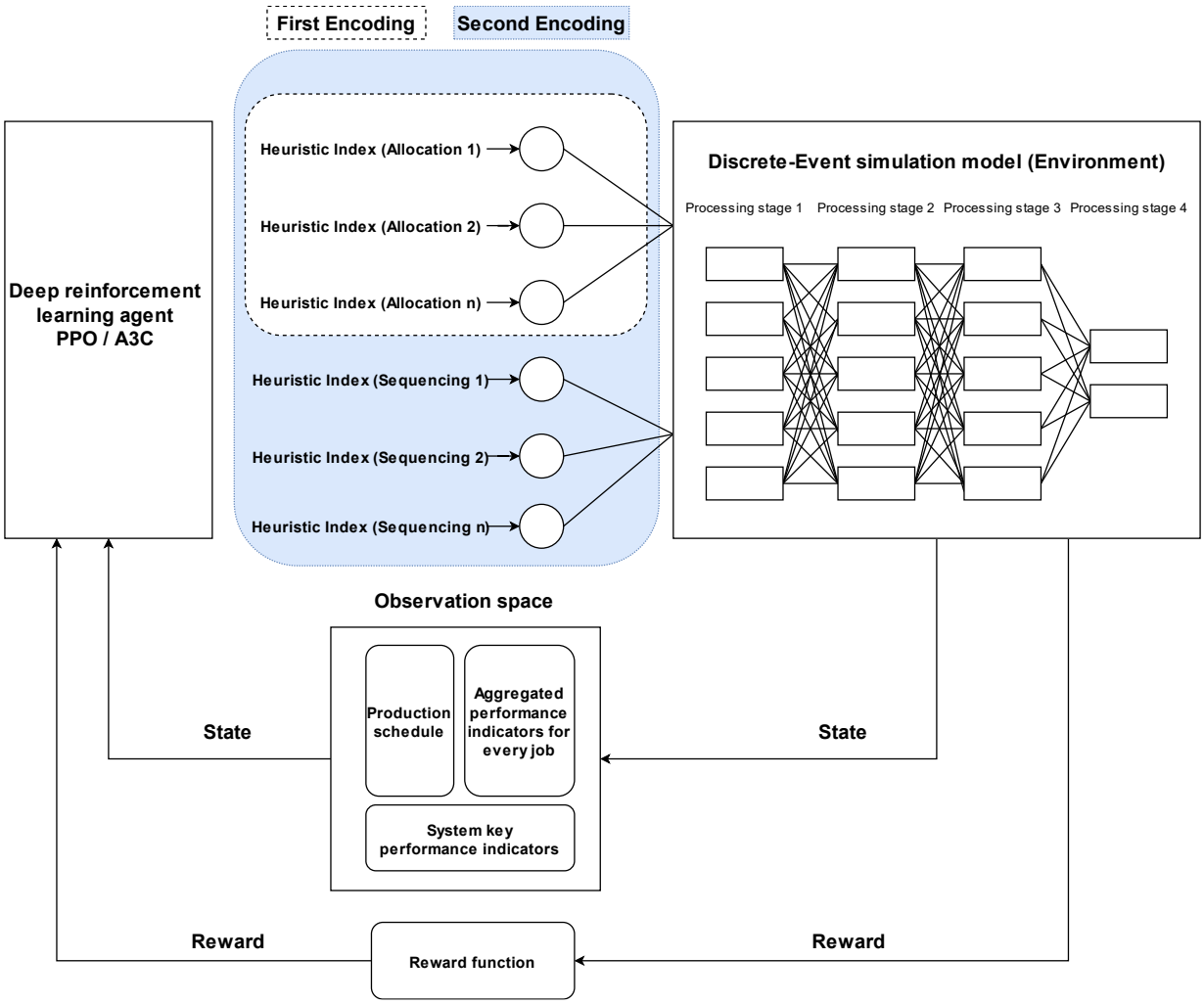

Figure 1: The conceptual design of the proposed approach.

4. Experiments and results

The experiments are designed to investigate the performance of the Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C) for addressing the investigated problem and test the suggested encoding. The encoding of the problem is an attempt to represent complex industrial problems as game-like environments for Deep Reinforcement Learning (DRL) algorithms. We conducted experiments on two encodings. The first encoding allows agents to control the selection of allocation algorithms in the environment over time. The second encoding allows agents to control the selection of allocation and sequencing algorithms for solving the considered problem.

In this work, we relied on Ray RLlib [40] framework for applying the suggested DRL algorithms. Ray is an open-source framework that allows to investigate, implement, and evaluate DRL algorithms in a highly parallelized manner. The goal is to achieve an effective distribution of computations across different workers to leverage the capabilities of modern hardware for rapid experiments.

The investigated algorithms are configured with the discount factor 0.05 so that the agent seeks to maximize the immediate reward, as this is the desired behavior in our experimental setup. Additionally, preliminary tuning of the algorithm's parameters suggests the significant importance of the learning rate for the agent's learning performance. A learning rate of 0.0001 shows to result in the desired behavior. The experiments are conducted on three problem instances of distinct complexity. Agents start solving problems of medium complexity, followed by a simpler problem, then concluding with a hard problem. The complexities of the problems are empirically identified using the same metaheuristic algorithm for solving them. We measured the average required computational effort of the metaheuristic to coverage.

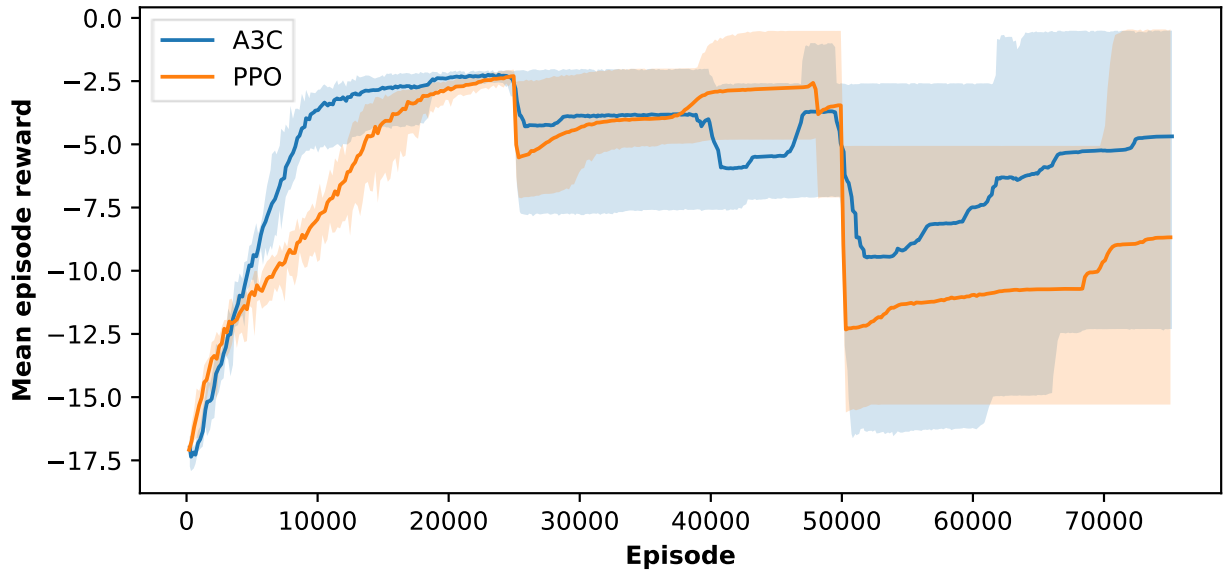

Figure 2: The computational results on the performance of A3C and PPO in the first encoding.

The goal is eventually to investigate the behavior of the DRL agents for solving different problems of different complexity. The obtained computational results of the conducted experiments are presented in Figure 2 and Figure 3. Particularly important is to observe the agent's adaptation to the changing nature of the jobs that must be scheduled in the investigated production system. The computational results of the first encoding are presented in Figure 2. The PPO and A3C are employed to select different allocation algorithms over time for the allocation and reallocation of jobs between available machines for solving the problem. These selections are then used in the simulation environment. The simulation component is developed using the Salabim simulation package [41]. Salabim is an open-source discrete-event simulation package that allows modeling discrete systems in python. The simulation model accesses different python classes, in which the logic of the allocation and sequencing algorithms is written. As shown in Figure 2, the steady increase in the mean reward on average suggests a successful learning process of the DRL agents for solving the problem.

We trained the agents for 25000 episodes for solving a problem instance. As seen in the figure, a drastic and sudden change in the performance of the algorithms can be observed every 25000, when we expose the agents to different problems. Of particular interest, is to observe whether the agents will recover and adjust their policies for solving different problems. Also, how long will the recovery process of different agents take? The obtained results show that agents can recover and find high-quality solutions for solving different problem instances of different complexity. On average, the recovery process of the A3C marginally outperforms the one of the PPO for dealing with the problems. The tendency of the algorithms performance recovery is a positive step in the context of applying DRL techniques for solving industrial problems. It implies that both algorithms, despite the demonstrated overall better performance of A3C, are able to adjust their learned policies to address different problems.

The computational results of the second encoding are presented in Figure 3. Briefly, this encoding allows DRL agents to control the selection of allocation and sequencing algorithms over time during a scheduling period for solving the problems. It implies that the agents control the allocation and reallocation processes of the jobs between available machines over time during a scheduling period. In addition, the action space of the agents is designed to allow them also to control the sequencing of jobs on the machines over time using different sequencing algorithms. This encoding is more sophisticated for DRL techniques than the previous one. The goal behind investigating this encoding is to evaluate the impact of complicating the encoding on the performance of the algorithm in terms of learning and recovery processes.

As shown in Figure 3, A3C and PPO are able to successfully learn to deal with the investigated problem. However, we can already notice the impact of complex multi-discrete action space on the performance of the algorithms for solving the problems. The results suggest that both algorithms require more time to reach a steady high mean-reward during the first 25000 episodes for solving the medium complicated problem.

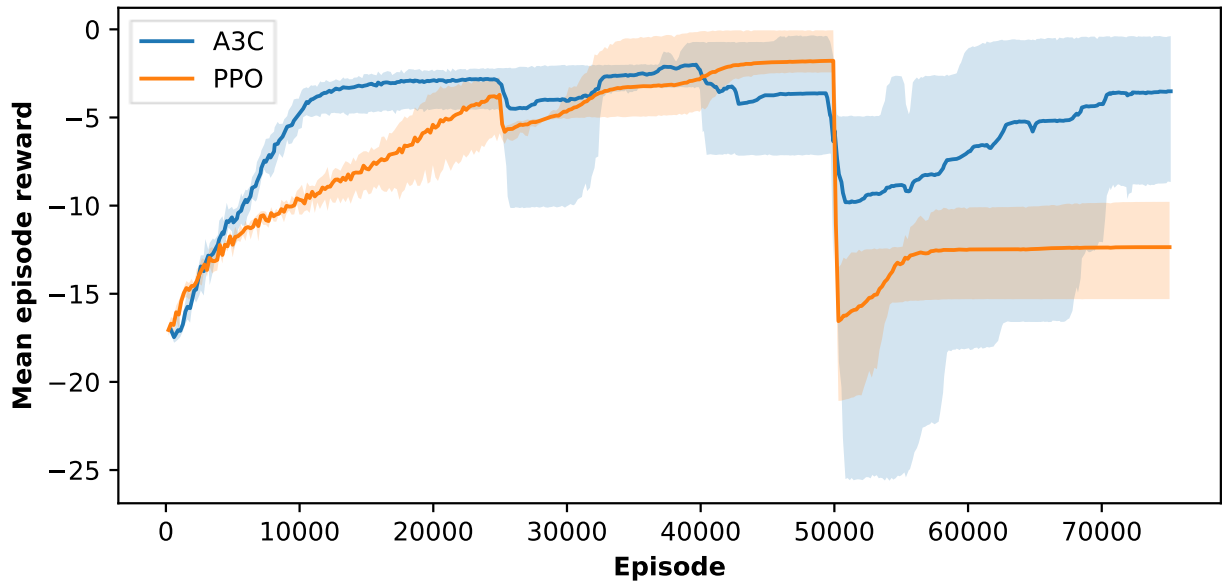

Figure 3: The computational results on the performance of A3C and PPO in the second encoding.

In addition, once again the outperformance of the A3C is more significant during the training process in terms of the obtained mean reward. In Figure 3, one can observe that the A3C agent (blue line) learn much faster than the PPO (orange line) during the first 25000 episodes. The recovery process of the A3C significantly outperforms the recovery process of the PPO after exposing the agent to different problems of different complexity. It implies that A3C is able to adjust achieved policy to address new problems much better than PPO. To increase the confidence in the obtained results, we decided to conduct further analysis on published problem instances in [42] that are based on [21]. The authors investigated a two-stage HFS scheduling problem with family major and minor setup times. The authors presented extensive analysis on the nature of the problem instance and presented an approach based on NeuroEvolution of Augmenting Topologies (NEAT) for solving the problems in [21]. Therefore, we built a simulation model for a two-stage production system and used the problem instance to further analyze the performance of the suggested approaches.

The computational results of the conducted experiments are presented in Table 1. We report here the best-found solutions by the DRL agents for solving the problem instances after collecting the results of a single run. The makespan C_{max} and the total tardiness are reported in minutes. The unit penalties and major setups are reported in number. The A3C achieves better results in terms of all perused objective values in comparison to NEAT and PPO for solving the first three problem instances. PPO on the other hand outperforms NEAT for solving two problem instances while NEAT achieves better performance in terms of minimizing the makespan. NEAT outperforms PPO and A3C for solving the fourth problem instance. The DRL agents show similar behavior for solving the two-stage problems. They achieve high-quality solutions for solving the first three problem instances and then decline after being exposed to the fourth problem instance. This demonstrates that DRL agents successfully adjust their policies for solving problems of presumably similar nature and necessitate more episodes to adjust their policy for solving the fourth.

Table 1: Computational results of PPO and A3C compared to NEAT [21].

	C_{max}	T	U	MS
Problem 1				
NEAT	17,768	124	0	114
PPO	18,609	0	0	54
A3C	**17,081**	**0**	**0**	**58**
Problem 2				
NEAT	20,916	303	0	149
PPO	17,874	0	0	55
A3C	**17,669**	**0**	**0**	**55**
Problem 3				
NEAT	20,584	0	0	142
PPO	18,228	0	0	**54**
A3C	**17,690**	0	0	56
Problem 4				
NEAT	**18,771**	**0**	**0**	113
PPO	19,970	2,245	4	52
A3C	19,767	1,627	4	52

We conducted another experiment and started to train the agents for solving the fourth problem instance and then switching to the others. The results of the DRL agents are significantly better than the reported results in the table. It implies that the nature of the fourth problem is marginally different than the others.

5. Conclusion and discussions

In this paper, we present a DRL approach that is coupled with a simulation environment to address HFS scheduling problems. We adopted two DRL algorithms that are presented by prominent practitioners: PPO and A3C originally developed by OpenAI and Google DeepMind, respectively. The conducted evaluation on the two-stage and four-stage HFS scheduling problems show that the DRL agents are able to address the problems using the presented formulation of the action space. The presented multi-discrete action space formulation allows us to expose the agent to the simulated production system and interact with it as a game. Both experiments show that exposing the agents to solve problems of different nature might lead to degradation in the performance until the agents adjust their policies. It implies that the generalization of the DRL agent's behavior is subject to further research.

Our results correspond to the generalization challenges discussed by Google Brain and DeepMind in [43]. To address this drawback two directions can be pursued. The first is to address how deep neural networks configurations affect the performance of the learning process [43]. The second direction is rather concentrated on their application. For instance, one can attempt to classify problem instances in terms of complexity. Different agents can be trained to deal with problems of similar nature to potentially achieve better results. Production systems daily will not result in scheduling problems of an entirely different nature. Some new jobs are released for scheduling and some finished. Therefore, sufficient training of a DRL agent might deliver solutions with sufficient quality for supporting instant decision-making.

6. References

[1] Baker, K.R. and D. Trietsch, Principles of Sequencing and Scheduling, Wiley-Blackwell, Oxford, 2009.

[2] Pinedo, M.L., Scheduling: Theory, Algorithms, and Systems, Springer New York, 2012.

[3] Ruiz, R. and J.A. Vázquez-Rodríguez, "The hybrid flow shop scheduling problem", European Journal of Operational Research, 205(1), 2010, pp. 1–18.

[4] Conway, R.W., W.L. Maxwell, and L.W. Miller, Theory of Scheduling, Addison-Wesley Pub. Co, 1967.

[5] Lenstra, J.K., A. Rinnooy Kan, and P. Brucker, "Complexity of Machine Scheduling Problems", in Studies in Integer Programming, P.L. Hammer, E.L. Johnson, B.H. Korte and G.L. Nemhauser, Editors. 1977. Elsevier.

[6] Gupta, J.N.D., "Two-Stage, Hybrid Flowshop Scheduling Problem", The Journal of the Operational Research Society, 39(4), 1988, p. 359.

[7] Voß, S., "The Two — Stage Hybrid — Flowshop Scheduling Problem with Sequence — Dependent Setup Times", in Operations Research in Production Planning and Control, G. Fandel, T. Gulledge, and A. Jones, Editors. 1993. Springer Berlin Heidelberg.

[8] Neufeld, J.S., J.N. Gupta, and U. Buscher, "A comprehensive review of flowshop group scheduling literature", Computers & Operations Research, 70, 2016, pp. 56–74.

[9] Ribas, I., R. Leisten, and J.M. Framiñan, "Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective", Computers & Operations Research, 37(8), 2010, pp. 1439–1454.

[10] Hunsucker, J.L. and J.R. Shah, "Comparative performance analysis of priority rules in a constrained flow shop with multiple processors environment", European Journal of Operational Research, 72(1), 1994, pp. 102–114.

[11] Nahhas, A., P. Aurich, T. Reggelin, and J. Tolujew, "Heuristic and Metaheuristic Simulation-Based Optimization for Solving a Hybrid Flow Shop Scheduling Problem", in The 15th International Conference on Modeling and Applied Simulation, A. G. Bruzzone, F. De Felice, C. Frydman, M. Massei, Y. Merkuryev, and A. Solis, Editors. 2016: RENDE (CS), ITALY.

[12] Holland, J.H., Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence / by John H.Holland, University of Michigan Press, Ann Arbor, Mich., 1975.

[13] Kirkpatrick, S., C.D. Gelatt, and M.P. Vecchi, "Optimization by simulated annealing", Science (New York, N.Y.), 220(4598), 1983, pp. 671–680.

[14] Liao, C.-J., C.-T. Tseng, and P. Luarn, "A discrete version of particle swarm optimization for flowshop scheduling problems", Computers & Operations Research, 34(10), 2007, pp. 3099–3111.

[15] Gómez-Gasquet, P., C. Andrés, and F.-C. Lario, "An agent-based genetic algorithm for hybrid flowshops with sequence dependent setup times to minimise makespan", Expert Systems with Applications, 39(9), 2012, pp. 8095–8107.

[16] Lei, D. and Y. Zheng, "Hybrid flow shop scheduling with assembly operations and key objectives: A novel neighborhood search", Applied Soft Computing, 61, 2017, pp. 122–128.

[17] Ross, P., "Hyper-Heuristics", in Search Methodologies, E.K. Burke and G. Kendall, Editors. 2005. Springer US: Boston, MA.

[18] Aurich, P., A. Nahhas, T. Reggelin, and J. Tolujew, "Simulation-based Optimization for Solving a Hybrid

Flow Shop Scheduling Problem", in Proceedings of the 2016 Winter Simulation Conference. 2016. IEEE Press: Piscataway, NJ, USA.
[19] Xu, W.-J., L.-J. He, and G.-Y. Zhu, "Many-objective flow shop scheduling optimisation with genetic algorithm based on fuzzy sets", International Journal of Production Research, 59(3), 2021, pp. 702–726.
[20] Nahhas, A., M. Krist, and K. Turowski, "An adaptive scheduling framework for solving multi-objective hybrid flow shop scheduling problems", in Proceedings of the 54th Hawaii International Conference on System Sciences, T. Bui, Editor, HICSS. 2021.
[21] Lang, S., T. Reggelin, F. Behrendt, and A. Nahhas, "Evolving Neural Networks to Solve a Two-Stage Hybrid Flow Shop Scheduling Problem with Family Setup Times", in Proceedings of the 53rd Hawaii International Conference on System Sciences, T. Bui, Editor, HICSS. 2020.
[22] Nahhas, A., S. Lang, S. Bosse, and K. Turowski, "Toward Adaptive Manufacturing: Scheduling Problems in the Context of Industry 4.0", in 2018 Sixth International Conference on Enterprise Systems (ES), 2018 Sixth International Conference on Enterprise Systems (ES). 2018.
[23] Papadimitriou, C.H. and J.N. Tsitsiklis, "The Complexity of Markov Decision Processes", Mathematics of Operations Research, 12(3), 1987, pp. 441–450.
[24] LeCun, Y., Y. Bengio, and G. Hinton, "Deep learning", Nature, 521(7553), 2015, pp. 436–444.
[25] Arulkumaran, K., M.P. Deisenroth, M. Brundage, and A.A. Bharath, "Deep Reinforcement Learning: A Brief Survey", IEEE Signal Processing Magazine, 34(6), 2017, pp. 26–38.
[26] Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, Proximal Policy Optimization Algorithms, 20/07/2017.
[27] Sutton, R.S., D. McAllester, S. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation", Advances in Neural Information Processing Systems, 12, 1999.
[28] Schulman, J., S. Levine, P. Moritz, M.I. Jordan, and P. Abbeel, Trust Region Policy Optimization, 19/02/2015.
[29] Mnih, V., A.P. Badia, M. Mirza, A. Graves, T.P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning", ICML.
[30] Watkins, C.J. and P. Dayan, "Technical Note: Q-Learning", Machine Learning, 8(3), 1992, pp. 279–292.
[31] Mnih, V., K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning", Nature, 518(7540), 2015, pp. 529–533.
[32] Kanervisto, A., C. Scheller, and V. Hautamäki, Action Space Shaping in Deep Reinforcement Learning, 02/04/2020.
[33] Han, Guo, and Su, "A Reinforcement Learning Method for a Hybrid Flow-Shop Scheduling Problem", Algorithms, 12(11), 2019, p. 222.
[34] Ren, J., C. Ye, and F. Yang, "Solving flow-shop scheduling problem with a reinforcement learning algorithm that generalizes the value function with neural network", Alexandria Engineering Journal, 60(3), 2021, pp. 2787–2800.
[35] Wang, J., S. Qu, J. Wang, J.O. Leckie, and R. Xu, "Real-Time Decision Support with Reinforcement Learning for Dynamic Flowshop Scheduling", in Smart SysTech 2017; European Conference on Smart Objects, Systems and Technologies. 2017.
[36] Lang, S., F. Behrendt, N. Lanzerath, T. Reggelin, and M. Muller, "Integration of Deep Reinforcement Learning and Discrete-Event Simulation for Real-Time Scheduling of a Flexible Job Shop Production", in 2020 Winter Simulation Conference (WSC), 2020 Winter Simulation Conference (WSC), Orlando, FL, USA, 12/14/2020 - 12/18/2020. IEEE.
[37] Zhu, J., H. Wang, and T. Zhang, "A Deep Reinforcement Learning Approach to the Flexible Flowshop Scheduling Problem with Makespan Minimization", in 2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS), 2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS), Liuzhou, China, 11/20/2020 - 11/22/2020. IEEE.
[38] Li, S., "A hybrid two-stage flowshop with part family, batch production, major and minor set-ups", European Journal of Operational Research, 102(1), 1997, pp. 142–156.
[39] Delalleau, O., M. Peter, E. Alonso, and A. Logut, Discrete and Continuous Action Representation for Practical RL in Video Games, 23/12/2019.
[40] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica, "RLlib: Abstractions for Distributed Reinforcement Learning", International Conference on Machine Learning, 2018, pp. 3053–3062.
[41] van der Ham, R., "Salabim: open source discrete event simulation and animation in python", in Proceedings of the 2018 Winter Simulation Conference. 2018.
[42] Lang, S., T. Reggelin, J. Schmidt, M. Müller, and A. Nahhas, "NeuroEvolution of augmenting topologies for solving a two-stage hybrid flow shop scheduling problem: A comparison of different solution strategies", Expert Systems with Applications, 172, 2021, p. 114666.
[43] Zhang, C., O. Vinyals, R. Munos, and S. Bengio, A Study on Overfitting in Deep Reinforcement Learning, 18/04/2018.